Package 'estimatr'

October 10, 2026

Type Package

Title Fast Estimators for Design-Based Inference

Version 2.0.1

Description Fast procedures for a small set of commonly-used, design-appropriate estimators with robust standard errors and confidence intervals. Includes estimators for linear regression, instrumental variables regression, difference-in-means, Horvitz-Thompson estimation, and regression improving precision of experimental estimates by interacting treatment with centered pre-treatment covariates introduced by Lin (2013) <doi:10.1214/12-AOAS583>. Fixed effects are absorbed by alternating projections rather than by dummy expansion, and Horvitz-Thompson variance is computed from the randomization declaration.

URL https://declaredesign.org/r/estimatr/, https://github.com/DeclareDesign/estimatr

BugReports https://github.com/DeclareDesign/estimatr/issues

License MIT + file LICENSE

Encoding UTF-8

Depends R (>= 3.6.0)

Imports Formula, generics, methods, Rcpp, rlang, stats, tibble

Suggests AER, car, clubSandwich, DeclareDesign, dplyr, emmeans, estimability, ivreg, knitr, modelsummary, randomizr, rmarkdown, sandwich, testthat (>= 3.0.0), texreg

VignetteBuilder knitr

LinkingTo Rcpp, RcppEigen

Config/testthat/edition 3

Config/roxygen2/version 8.0.0

NeedsCompilation yes

Author Alexander Coppock [aut, cre], Graeme Blair [aut], Jasper Cooper [aut],

Luke Sonnet [aut],
Macartan Humphreys [ctb],
Neal Fultz [ctb],
Lily Medina [ctb],
Russell Lenth [ctb],
Molly Offer-Westort [ctb]

Maintainer Alexander Coppock <acoppock@gmail.com>

Repository CRAN

Date/Publication 2026-10-10 07:10:02 UTC

Contents

estimatr-package 2
augment.lm_robust 3
difference_in_means 4
estimatr-defunct 6
estimatr_tidiers 7
extract.lm_robust 8
horvitz_thompson 10
iv_robust 12
lh_robust 14
lm_lin 15
lm_robust 18
lm_robust_fit 21
predict.lm_robust 23

Index **26**

`estimatr-package` *estimatr: Fast Estimators for Design-Based Inference*

Description

Fast procedures for a small set of commonly-used, design-appropriate estimators with robust standard errors and confidence intervals. Provides `lm_robust`, `lm_lin`, `iv_robust`, `lh_robust`, `difference_in_means`, and `horvitz_thompson`, with feols-style fixed effects absorption and design-aware Horvitz-Thompson variance. See `vignette("estimatr2.0")` for what changes in version 2.0 and what does not.

Author(s)

Maintainer: Alexander Coppock <acoppock@gmail.com>

Authors:

- Alexander Coppock <acoppock@gmail.com>
- Graeme Blair <graeme.blair@gmail.com>
- Jasper Cooper <jjc2247@columbia.edu>

- Luke Sonnet <luke.sonnet@gmail.com>

Other contributors:

- Macartan Humphreys <macartan@gmail.com> [contributor]
- Neal Fultz <nfultz@gmail.com> [contributor]
- Lily Medina <lilymiru@gmail.com> [contributor]
- Russell Lenth <russell-lenth@uiowa.edu> [contributor]
- Molly Offer-Westort <mollyow@uchicago.edu> [contributor]

See Also

Useful links:

- https://declaredesign.org/r/estimatr/
- https://github.com/DeclareDesign/estimatr
- Report bugs at https://github.com/DeclareDesign/estimatr/issues

---

`augment.lm_robust` *Augment a Model Object with Fitted Values and Residuals*

Description

Returns the model frame with `.fitted` and `.resid` columns appended, the form downstream packages expect from `broom::augment()`. Supplying `newdata` returns that instead, with `.fitted` only.

Usage

```
## S3 method for class 'lm_robust'
augment(x, data = NULL, newdata = NULL, ...)

## S3 method for class 'iv_robust'
augment(x, data = NULL, newdata = NULL, ...)
```

Arguments

`x`	An `lm_robust` or `iv_robust` object.
`data`	The data to augment, defaulting to the model frame.
`newdata`	Optional new data to predict on instead.
`...`	(optional) Ignored.

Value

A `data.frame`.

Examples

```
set.seed(55)
dat <- data.frame(x = rnorm(50), z = rep(0:1, 25))
dat$y <- dat$x + 0.4 * dat$z + rnorm(50)
fit <- lm_robust(y ~ x + z, data = dat)

head(augment(fit))

# Supplying newdata returns predictions on it, with .fitted only
head(augment(fit, newdata = dat[1:5, ]))
```

---

difference_in_means *Design-Based Difference-in-Means Estimator*

Description

Estimates an average treatment effect as a difference in means, choosing the point estimator, the variance, and the degrees of freedom that match the randomization rather than requiring you to name them. Unit, cluster, blocked, block-cluster, matched-pair, and matched-pair cluster designs are recognised, and the `design` element of the result reports which case applied.

Usage

```
difference_in_means(
  formula,
  data,
  blocks,
  clusters,
  weights,
  subset,
  se_type = c("default", "none"),
  condition1 = NULL,
  condition2 = NULL,
  ci = TRUE,
  alpha = 0.05
)
```

Arguments

`formula`	(required) An object of class formula with one variable on the RHS
`data`	(optional) A `data.frame`
`blocks`	(optional) A bare (unquoted) name of the block variable
`clusters`	(optional) A bare (unquoted) name of the cluster variable
`weights`	(optional) The bare (unquoted) name of the weights variable
`subset`	(optional) A bare (unquoted) expression specifying a subset

`se_type`	(optional) `"default"` or `"none"`
`condition1`	(optional) Value in treatment for the control condition
`condition2`	(optional) Value in treatment for the treatment condition
`ci`	(optional) Logical. Whether to compute p-values and confidence intervals.
`alpha`	(optional) The significance level, 0.05 by default.

Details

Selects the appropriate point estimate, standard errors, and degrees of freedom for unit randomized, cluster randomized, block randomized, block-cluster randomized, matched-pairs, and matched-pair cluster randomized designs.

Blocks of different sizes. For unit randomized blocks, blocks are classified by how many units each arm holds rather than by how large the block is. A block with at least two treated and two control units has its own Neyman variance. A block with a single treated or single control unit has no estimable within-block variance, so the variation across such blocks stands in for it. A design containing both kinds combines the two parts by squared share of the sample, following Pashley and Miratrix (2021). The `design` element of the returned object reports which case applied: `"Blocked"`, `"Matched-pair"`, `"Small blocks"`, or `"Hybrid blocked"`.

Two designs are refused, because the variance genuinely cannot be estimated: exactly one block with a singleton arm, and a set of different-sized such blocks in which one holds half or more of their units. Both messages suggest merging blocks or using `lm_robust()` with block fixed effects.

If weights are specified, estimation is handed to `lm_robust()` with HC2 standard errors.

Blocks of clusters. Pashley and Miratrix treat treatment assigned within blocks, not blocks of clusters, so blocked designs with `clusters` use the earlier estimators. Every block must have at least two treated and two control clusters, unless the design is matched-pair clustered, where the variance is estimated across blocks. A block with a single treated or control cluster is refused: its within-block variance is not estimable, and estimating it anyway understates the standard error by roughly the block's cluster count.

Value

An object of class `"difference_in_means"`, a list holding `coefficients`, `std.error`, `df`, `statistic`, `p.value`, `conf.low`, `conf.high`, `term`, `outcome`, `condition1`, `condition2`, `vcov`, `nobs`, `alpha`, `call`, and `design`, a string naming the case that applied: `"Standard"`, `"Blocked"`, `"Small blocks"`, `"Hybrid blocked"`, `"Matched-pair"`, `"Clustered"`, `"Block-clustered"`, or `"Matched-pair clustered"`.

References

Gerber, Alan S. and Donald P. Green. 2012. *Field Experiments: Design, Analysis, and Interpretation*. New York: W.W. Norton.

Imai, Kosuke, Gary King, and Clayton Nall. 2009. "The Essential Role of Pair Matching in Cluster-Randomized Experiments." *Statistical Science* 24(1): 29-53. doi:10.1214/08STS274.

Pashley, Nicole E. and Luke W. Miratrix. 2021. "Insights on Variance Estimation for Blocked and Matched Pairs Designs." *Journal of Educational and Behavioral Statistics* 46(3): 271-296. doi:10.3102/1076998620946272.

Examples

```
set.seed(30)
dat <- data.frame(y = rnorm(100), z = rep(0:1, 50))

# Unblocked, unclustered: the Welch-corrected two-sample difference
fit <- difference_in_means(y ~ z, data = dat)
fit
fit$design

# Blocked designs use the Neyman variance within each block
dat_bl <- data.frame(
  bl = rep(1:10, each = 10),
  z  = rep(rep(0:1, each = 5), times = 10)
)
dat_bl$y <- rnorm(100) + 0.3 * dat_bl$z
difference_in_means(y ~ z, data = dat_bl, blocks = bl)

# Matched pairs are recognised as such
dat_pr <- data.frame(pr = rep(1:50, each = 2), z = rep(c(0, 1), 50))
dat_pr$y <- rnorm(100) + 0.3 * dat_pr$z
difference_in_means(y ~ z, data = dat_pr, blocks = pr)$design

# Blocks of unequal shape, which earlier versions refused, use the
# Pashley and Miratrix (2021) estimators. `design` reports which case
# applied rather than leaving it to be inferred from the block sizes.
dat_hy <- rbind(dat_bl[c("bl", "z", "y")],
                transform(dat_pr[c("pr", "z", "y")], bl = pr + 100)[c("bl", "z", "y")])
difference_in_means(y ~ z, data = dat_hy, blocks = bl)$design

# Clustered assignment
dat_cl <- data.frame(cl = rep(1:20, each = 5))
dat_cl$z <- rep(rep(0:1, each = 5), times = 10)
dat_cl$y <- rnorm(100) + 0.3 * dat_cl$z
difference_in_means(y ~ z, data = dat_cl, clusters = cl)
```

`estimatr-defunct` *Functions removed in 2.0*

Description

`commarobust()` and `starprep()` were helpers for producing robust standard errors outside the package's own estimators and formatting them for **stargazer**. Both are removed.

Usage

```
commarobust(...)

starprep(...)
```

Arguments

...	(optional) Ignored.

Details

They are kept here as names that error rather than deleted outright, so that a script written against estimatr 1.x says what happened and what to do instead of failing with `could not find function`.

`commarobust()` recomputed robust standard errors on a fitted `lm`. Fit the model with `lm_robust()` instead, which is what it was reimplementing.

`starprep()` prepared a list of fits for **stargazer**, which has not been maintained for years. Table-building now goes through **modelsummary**, which reads `tidy()` and `glance()` and therefore works on every estimator in this package without any adapter.

Value

Never returns; both functions signal an error.

Examples

```
# Both of these error. The replacements:
set.seed(1)
dat <- data.frame(y = rnorm(20), z = rep(0:1, 10))

# was: commarobust(lm(y ~ z, data = dat))
lm_robust(y ~ z, data = dat)

# was: starprep(fit1, fit2) |> stargazer::stargazer()
# now: modelsummary::modelsummary(list(fit1, fit2))
```

`estimatr_tidiers` *Tidy an estimatr object*

Description

Tidy an estimatr object

Usage

```
## S3 method for class 'lm_robust'
tidy(x, conf.int = TRUE, conf.level = NULL, ...)

## S3 method for class 'iv_robust'
tidy(x, conf.int = TRUE, conf.level = NULL, ...)

## S3 method for class 'difference_in_means'
tidy(x, conf.int = TRUE, conf.level = NULL, ...)
```

```
## S3 method for class 'horvitz_thompson'
tidy(x, conf.int = TRUE, conf.level = NULL, ...)

## S3 method for class 'lh_robust'
tidy(x, conf.int = TRUE, conf.level = NULL, ...)

## S3 method for class 'lh'
tidy(x, conf.int = TRUE, conf.level = NULL, ...)
```

Arguments

x	An object returned by one of the estimators
conf.int	Logical, whether to include confidence intervals.
conf.level	The confidence level for intervals.
...	(optional) Ignored.

Value

A tibble with one row per term (and per outcome, for a multivariate fit): `term`, `estimate`, `std.error`, `statistic`, `p.value`, `conf.low`, `conf.high`, `df`, and `outcome`. A tibble rather than a plain data frame, as broom's tidiers return; 1.x returned a data frame.

Examples

```
set.seed(50)
dat <- data.frame(x = rnorm(50), z = rep(0:1, 25))
dat$y <- dat$x + 0.4 * dat$z + rnorm(50)
fit <- lm_robust(y ~ x + z, data = dat)

# One row per term, with the interval the fit was built with
tidy(fit)
tidy(fit, conf.int = FALSE)
tidy(fit, conf.level = 0.9)

# The same shape for every estimator in the package
tidy(difference_in_means(y ~ z, data = dat))
```

---

extract.lm_robust *Extract model data for the* **texreg** *package*

Description

Prepares an `lm_robust` or `iv_robust` fit for **texreg**. Largely a clone of texreg's own `extract.lm` method.

Usage

```
extract.lm_robust(
  model,
  include.ci = TRUE,
  include.rsquared = TRUE,
  include.adjrs = TRUE,
  include.nobs = TRUE,
  include.fstatistic = FALSE,
  include.rmse = TRUE,
  include.nclusts = TRUE,
  ...
)

extract.iv_robust(
  model,
  include.ci = TRUE,
  include.rsquared = TRUE,
  include.adjrs = TRUE,
  include.nobs = TRUE,
  include.fstatistic = FALSE,
  include.rmse = TRUE,
  include.nclusts = TRUE,
  ...
)
```

Arguments

`model`	An `lm_robust` or `iv_robust` fit.
`include.ci`, `include.rsquared`, `include.adjrs`, `include.nobs`	Logical.
`include.fstatistic`, `include.rmse`, `include.nclusts`	Logical.
`...`	(optional) Ignored.

Details

These are exported as plain functions rather than registered with `S3method()` because that is how **texreg** finds them: it looks up `extract.<class>` by name in the package namespace rather than dispatching on a generic it owns. Registering them the usual way would leave texreg unable to see them.

texreg is the only consumer. Table building through **modelsummary** needs nothing here, since it reads `tidy()` and `glance()` and so already works on every estimator in this package.

Value

A **texreg** object.

Examples

```
set.seed(60)
dat <- data.frame(x = rnorm(50), z = rep(0:1, 25))
dat$y <- dat$x + 0.4 * dat$z + rnorm(50)
fit <- lm_robust(y ~ x + z, data = dat)

if (requireNamespace("texreg", quietly = TRUE)) {
  texreg::screenreg(fit)
}
```

`horvitz_thompson` *Horvitz-Thompson Estimator with Inverse Probability Weighting*

Description

Estimates treatment effects via inverse probability weighting when treatment assignment probabilities are known. Supports all `randomizr` designs as well as arbitrary designs supplied via a permutation matrix.

Usage

```
horvitz_thompson(
  formula,
  data,
  condition_prs = NULL,
  condition1 = NULL,
  condition2 = NULL,
  se_type = "youngs",
  ci = TRUE,
  alpha = 0.05
)
```

Arguments

`formula`	(required) A formula `Y ~ Z`, with one outcome.
`data`	(optional) A `data.frame` with one row per unit of the design.
`condition_prs`	(required) Treatment probability specification. One of: • An `ra_declaration` from `randomizr`: **strongly preferred.** All standard designs (simple/Bernoulli, complete, blocked, clustered, blocked-and-clustered, and arbitrary permutation matrices) are supported, and the variance estimator uses exact design-aware joint inclusion probabilities. Any design for which you know the block structure, cluster structure, marginal treatment probabilities, and whether randomization is simple or complete can be expressed as `declare_ra(blocks = bl, clusters = cl, prob = pi, simple = FALSE)`. There is no parametric design that requires the alternatives below. For fully custom designs, use `declare_ra(permutation_matrix = perm)`.

- A named numeric vector of marginal condition probabilities, e.g. `c("0" = 0.4, "1" = 0.6)`. Uses the conservative Young's simple-randomization variance bound, which is valid for any design but exact only for Bernoulli (simple) randomization. For complete or blocked designs this overstates uncertainty; use an `ra_declaration` to get the tighter design-aware variance.
- A two-column matrix or data frame of per-unit probabilities with columns named by condition labels. Same conservative Young's bound as the named vector above.

`condition1` (optional) Label of the control condition (first sorted condition by default).

`condition2` (optional) Label of the treatment condition (second sorted condition by default).

`se_type` (optional) `"youngs"` (default) or `"none"`.

`ci` (optional) Logical; whether to compute p-values and confidence intervals.

`alpha` (optional) Significance level, 0.05 by default.

Details

With more than two arms, `condition1` and `condition2` select the contrast, and the estimand remains the average treatment effect over all N units the design covers. The estimator therefore divides by N, not by the number of units landing in the two conditions, and the variance uses the joint assignment probabilities implied by the arm sizes. `data` must hold one row per unit of the design, in the design's order, including units assigned to arms outside the contrast.

Value

An object of class `"horvitz_thompson"` with fields `coefficients`, `std.error`, `statistic`, `p.value`, `conf.low`, `conf.high`, `df`, `nobs` (the number of units in the design, including any arms outside the contrast), `vcov`, `se_type`, `condition1`, `condition2`, `outcome`, and `term`.

Examples

```
set.seed(40)
dat <- data.frame(y = rnorm(100), z = rep(0:1, 50))

# A named vector of condition probabilities gives the conservative
# simple-randomization bound, valid for any design
horvitz_thompson(y ~ z, data = dat, condition_prs = c("0" = 0.5, "1" = 0.5))

# Passing the randomization declaration instead is what buys the
# design-aware variance, and it is the recommended form
if (requireNamespace("randomizr", quietly = TRUE)) {
  decl <- randomizr::declare_ra(N = 100, m = 50)
  dat$z2 <- randomizr::conduct_ra(decl)
  print(horvitz_thompson(y ~ z2, data = dat, condition_prs = decl))

  # Blocked and clustered designs need no extra arguments: the declaration
  # already carries the structure
  bl <- rep(1:4, each = 25)
  decl_bl <- randomizr::declare_ra(blocks = bl, prob = 0.4)
```

```
  dat$z3 <- randomizr::conduct_ra(decl_bl)
  print(horvitz_thompson(y ~ z3, data = dat, condition_prs = decl_bl))

  # Any two arms of a multi-arm design can be contrasted, with the estimand
  # still defined over all N units
  decl3 <- randomizr::declare_ra(N = 100, conditions = c("a", "b", "c"))
  dat$z4 <- randomizr::conduct_ra(decl3)
  print(horvitz_thompson(y ~ z4, data = dat, condition_prs = decl3,
                         condition1 = "a", condition2 = "c"))
}
```

`iv_robust` *Two-Stage Least Squares Instrumental Variables Regression*

Description

Fits a two-stage least squares instrumental variables regression and returns heteroskedasticity-robust or cluster-robust standard errors, with optional weak-instrument, Wu-Hausman, and overidentification diagnostics.

Usage

```
iv_robust(
  formula,
  data,
  weights,
  subset,
  clusters,
  fixed_effects,
  se_type = NULL,
  ci = TRUE,
  alpha = 0.05,
  diagnostics = FALSE,
  return_vcov = TRUE,
  try_cholesky = FALSE
)
```

Arguments

`formula`	(required) An object of class formula with regressors and instruments, e.g. `y ~ x1 + x2 | z1 + z2`.
`data`	(optional) A `data.frame`
`weights`	(optional) The bare (unquoted) name of the weights variable
`subset`	(optional) A bare (unquoted) expression specifying a subset
`clusters`	(optional) A bare (unquoted) name of the cluster variable

`fixed_effects` (optional) A one-sided formula of fixed effects to absorb, such as `~ blockID`. Uses FWL demeaning (see `lm_robust()` for details and SE type restrictions). Diagnostics are not available with `fixed_effects`.

`se_type` (optional) The standard error type. `"HC2"` and `"HC3"` work with `fixed_effects` at any number of factors: the second stage runs on fitted regressors, but those are demeaned by the same fixed effects, so the leverage decomposition `lm_robust()` describes applies unchanged. `"CR2"` with `fixed_effects` expands the dummies, as in estimatr 1.0.6. Defaults: `"HC2"` (no clusters, with or without FE), `"CR2"` (clusters, no FE), `"CR0"` (clusters, with FE).

`ci` (optional) Logical. Whether to compute p-values and confidence intervals.

`alpha` (optional) The significance level, 0.05 by default.

`diagnostics` (optional) Logical. Whether to compute IV diagnostic statistics: the first-stage F test of the excluded instruments for each endogenous regressor, a regression-based Wu-Hausman test of endogeneity, and, when the model is overidentified, a test of the overidentifying restrictions. That test is Sargan's with `se_type = "classical"` and Wooldridge's (1995) robust score test otherwise, with the score's variance summed within clusters when `clusters` is given. With `weights`, each test is the one on the model with every row multiplied by the square root of its weight. The first-stage F and Wu-Hausman tests are Wald tests under the fit's own `se_type`, so a classical one is valid exactly when the classical weighted standard errors are and a robust one exactly when the robust ones are. The robust score test uses the score's HC0 or CR0 sandwich under every robust `se_type`, as Wooldridge (1995) and Stata define it; the HC1, HC2, HC3, CR2, and `"stata"` refinements correct a coefficient covariance and have no counterpart in a score test. The overidentification test is `NA`, with a warning, for a clustered fit with no more clusters than restrictions. All three reproduce Stata's `estat firststage`, `estat endogenous`, and `estat overid` on every row of the test suite's Stata fixture that Stata answers, except the robust score test after aweights under `forceweights`, where Stata computes the frequency-weight statistic instead.

`return_vcov` (optional) Logical. Whether to return the vcov matrix.

`try_cholesky` (optional) Logical. Whether to solve by Cholesky decomposition of `X'X` rather than by the default pivoted QR. `FALSE` by default, and worth turning on in most applied settings: about 1.4 times faster at n = 100,000 with two regressors, and 1.7 times faster at n = 200,000 with 60 regressors, where it is 0.15s against 0.25s. The saving is per fit, so it is worth most in a simulation that fits the same design thousands of times.

Rank deficiency is caught on either path. Redundant columns come back as `NA` exactly as they do from `lm()` whichever path ran, and a design that is rank deficient falls back to the QR.

Whether it is safe turns on one question, whether two regressors are nearly the same variable. Forming `X'X` squares the condition number, so the Cholesky path has about twice the rounding error of the QR, and only near-collinearity makes that visible. Differences of scale do not, because the columns are normalized before either decomposition, so a covariate in dollars beside one in years costs nothing. For a treatment indicator, a few covariates, block or cluster dummies, the centered interactions `lm_lin()` builds, or a factorial, the two paths agree to

at least 10 significant digits, which is why `difference_in_means()` sets it to `TRUE` internally. Agreement falls to about 3 digits as the scaled condition index reaches `1e6`, and the QR fallback takes over above roughly `1e8`. Nothing interpretable lives in that range: a design at `1e6` returns a coefficient of 4.8e4 with a standard error of 4.6e4 on a regressor whose true effect is zero. To check a design directly, scale the columns first, since the unscaled condition number of a design in mixed units is large for a reason that does not affect the fit: `kappa(sweep(X, 2, sqrt(colSums(X^2)), "/"), exact = TRUE)`.

Value

An object of class `"iv_robust"`, a list holding the estimate table in `coefficients`, `std.error`, `df`, `statistic`, `p.value`, `conf.low`, `conf.high`, `term`, and `outcome`; the fit in `fitted.values`, `residuals`, `vcov`, `nobs`, `k`, `rank`, `df.residual`, and `res_var`; the summary statistics `r.squared`, `adj.r.squared`, `tss`, and `fstatistic`; and `se_type`, `weighted`, `clustered`, `fes`, `alpha`, `terms`, `xlevels`, and `call`.

`residuals` are the structural residuals, `y - X beta`, rather than the second-stage ones. `ei.iv`, `terms_regressors`, and `formula` record the two-stage structure. With `diagnostics = TRUE` the object also holds `diagnostic_first_stage_fstatistic`, `diagnostic_endogeneity_test`, and `diagnostic_overid_test`.

Examples

```
set.seed(25)
n <- 200
dat <- data.frame(z = rbinom(n, 1, 0.5), cl = rep(1:20, each = 10))
dat$x <- dat$z * rbinom(n, 1, 0.7)
dat$y <- dat$x + rnorm(n)

# Endogenous regressor on the left of the bar, instrument on the right
fit <- iv_robust(y ~ x | z, data = dat)
tidy(fit)

# The same variance menu as lm_robust()
iv_robust(y ~ x | z, data = dat, se_type = "classical")
iv_robust(y ~ x | z, data = dat, clusters = cl)

# Weak-instrument, endogeneity, and overidentification tests
summary(iv_robust(y ~ x | z, data = dat, diagnostics = TRUE))
```

`lh_robust` *Linear Hypothesis Test for OLS with Robust Standard Errors*

Description

Tests a linear combination of coefficients, or several of them jointly, from a model fitted by `lm_robust()`. The robust variance and the degrees of freedom of the fit are carried through, so a clustered fit is tested on its cluster-adjusted degrees of freedom rather than on the residual ones.

Usage

```
lh_robust(..., data, linear_hypothesis)
```

Arguments

`...`	(optional) Other arguments passed to `lm_robust()`
`data`	(optional) A `data.frame`
`linear_hypothesis`	(required) A character string or matrix specifying the hypothesis, passed to `car::linearHypothesis`

Value

An object of class `"lh_robust"` with three components: `lm_robust`, the underlying fit; `lh`, one row per hypothesis holding `coefficients`, `std.error`, `statistic`, `p.value`, `alpha`, `conf.low`, `conf.high`, `df`, `term`, and `outcome`; and `joint_hypothesis`, the Wald F test of all of them at once, as `value`, `numdf`, `dendf`, and `p.value`. Under `se_type = "CR2"` each hypothesis's `df` is its own Satterthwaite approximation, as `clubSandwich::linear_contrast()` computes it, and `dendf` is the smallest of them.

Examples

```
set.seed(35)
dat <- data.frame(x = rnorm(100), z = rbinom(100, 1, 0.5),
                  cl = rep(1:10, each = 10))
dat$y <- dat$x + 0.5 * dat$z + rnorm(100)

# One linear combination of coefficients
fit <- lh_robust(y ~ x + z, data = dat, linear_hypothesis = "z + 2*x = 0")
fit
tidy(fit)

# Degrees of freedom follow the fit, so a clustered model tests against the
# cluster-adjusted df rather than the residual df
lh_robust(y ~ x + z, data = dat, clusters = cl,
          linear_hypothesis = "z + 2*x = 0")

# Several restrictions at once give one joint Wald test as well
joint <- lh_robust(y ~ x + z, data = dat, linear_hypothesis = c("x = 0", "z = 0"))
joint$joint_hypothesis
```

Description

Estimates an average treatment effect with covariate adjustment following Lin (2013): every covariate is centered, interacted with treatment, and entered alongside it. Centering is what makes the treatment coefficient the effect estimate, and the interactions avoid the bias Freedman (2008) identified in ordinary covariate-adjusted regression.

Usage

```
lm_lin(
  formula,
  covariates,
  data,
  weights,
  subset,
  clusters,
  se_type = NULL,
  ci = TRUE,
  alpha = 0.05,
  return_vcov = TRUE,
  try_cholesky = FALSE
)
```

Arguments

formula	(required) An object of class formula with only the treatment on the RHS
covariates	(required) A right-sided formula with pre-treatment covariates
data	(optional) A data.frame
weights	(optional) The bare (unquoted) name of the weights variable
subset	(optional) A bare (unquoted) expression specifying a subset
clusters	(optional) A bare (unquoted) name of the cluster variable
se_type	(optional) The sort of standard error (see lm_robust())
ci	(optional) Logical. Whether to compute p-values and confidence intervals.
alpha	(optional) The significance level, 0.05 by default.
return_vcov	(optional) Logical. Whether to return the vcov matrix.
try_cholesky	(optional) Logical. Whether to solve by Cholesky decomposition of X'X rather than by the default pivoted QR. FALSE by default, and worth turning on in most applied settings: about 1.4 times faster at n = 100,000 with two regressors, and 1.7 times faster at n = 200,000 with 60 regressors, where it is 0.15s against 0.25s. The saving is per fit, so it is worth most in a simulation that fits the same design thousands of times. Rank deficiency is caught on either path. Redundant columns come back as NA exactly as they do from lm() whichever path ran, and a design that is rank deficient falls back to the QR. A rank-deficient Lin design is worth reading carefully, because the treatment coefficient is the effect at the covariate means only while every covariate is interacted with the treatment. A duplicated or constant covariate loses its main

effect and its interaction together, and what is left is `lm_lin()` on the covariates that remain. A covariate that is constant within one treatment arm is different: the intercept, the treatment indicator, the centered covariate, and its interaction are then an exact four-column dependency, one of the four is dropped, and the fit is the same under any of those choices while the treatment coefficient is not. The effect at the covariate means is not identified there under any software, and no goodness-of-fit channel shows it: fitted values and `r.squared` agree to twelve digits across drop choices that move the treatment coefficient by a quarter of its size. The message raised at such a fit names the consequence as well as the dropped column, and a dropped treatment indicator, which leaves no treatment coefficient at all, is a warning rather than a message. An empty covariate-by-arm cell is usually a subgroup analysis run in a cell too thin to carry the covariate; collapsing that covariate's categories, or dropping it in that subgroup, is what restores the estimand.

Whether it is safe turns on one question, whether two regressors are nearly the same variable. Forming `X'X` squares the condition number, so the Cholesky path has about twice the rounding error of the QR, and only near-collinearity makes that visible. Differences of scale do not, because the columns are normalized before either decomposition, so a covariate in dollars beside one in years costs nothing. For a treatment indicator, a few covariates, block or cluster dummies, the centered interactions `lm_lin()` builds, or a factorial, the two paths agree to at least 10 significant digits, which is why `difference_in_means()` sets it to `TRUE` internally. Agreement falls to about 3 digits as the scaled condition index reaches `1e6`, and the QR fallback takes over above roughly `1e8`. Nothing interpretable lives in that range: a design at `1e6` returns a coefficient of 4.8e4 with a standard error of 4.6e4 on a regressor whose true effect is zero. To check a design directly, scale the columns first, since the unscaled condition number of a design in mixed units is large for a reason that does not affect the fit: `kappa(sweep(X, 2, sqrt(colSums(X^2)), "/"), exact = TRUE)`.

Value

An object of class `"lm_robust"`, as returned by `lm_robust()`, with two additions: `scaled_center`, the covariate means used for centering (taken after any function in the formula is evaluated), and `treatment_levels`. The treatment row of `coefficients` is the estimate of the average treatment effect.

References

Lin, Winston. 2013. "Agnostic Notes on Regression Adjustments to Experimental Data: Reexamining Freedman's Critique." The Annals of Applied Statistics 7(1): 295-318. doi:10.1214/12-AOAS583.

Examples

```
set.seed(20)
dat <- data.frame(
  x  = rnorm(40, mean = 2.3),
  x2 = rpois(40, lambda = 2),
```

```
  x3 = runif(40),
  z  = rep(0:1, 20),
  cl = rep(1:20, each = 2)
)
dat$y <- rnorm(40) + dat$x + 0.35 * dat$z

# lm_robust's interface plus one argument
fit <- lm_lin(y ~ z, covariates = ~ x, data = dat)
tidy(fit)

# Several covariates
lm_lin(y ~ z, covariates = ~ x + x2, data = dat)

# Covariates are centered after any function in the formula is evaluated
fit2 <- lm_lin(y ~ z, covariates = ~ x + log(x3), data = dat)
fit2$scaled_center["log(x3)"]
mean(log(dat$x3))

# Clusters, and multi-valued treatments whether or not they are factors
lm_lin(y ~ z, covariates = ~ x, data = dat, clusters = cl)
dat$z3 <- rep(1:3, length.out = 40)
lm_lin(y ~ z3, covariates = ~ x, data = dat)
lm_lin(y ~ factor(z3), covariates = ~ x, data = dat)

# Dropping the intercept gives the mean outcome under each condition
lm_lin(y ~ z3 - 1, covariates = ~ x, data = dat)
```

`lm_robust` *Ordinary Least Squares with Robust Standard Errors*

Description

Fits a linear model by ordinary least squares and returns heteroskedasticity-robust or cluster-robust standard errors, with the small-sample corrections used in design-based work. Fixed effects can be absorbed rather than expanded into dummy columns, at no cost in the available standard error types.

Usage

```
lm_robust(
  formula,
  data,
  weights,
  subset,
  clusters,
  fixed_effects,
  se_type = NULL,
  ci = TRUE,
  alpha = 0.05,
```

```
  return_vcov = TRUE,
  try_cholesky = FALSE
)
```

Arguments

formula	(required) An object of class formula, as in `lm()`
data	(optional) A `data.frame`
weights	(optional) The bare (unquoted) name of the weights variable
subset	(optional) A bare (unquoted) expression specifying a subset
clusters	(optional) A bare (unquoted) name of the cluster variable
fixed_effects	(optional) A one-sided formula of fixed effects to absorb rather than expand into dummy columns, such as `~ blockID` or `~ block + year`. Each variable is demeaned within the groups before OLS is run, so by the Frisch-Waugh-Lovell theorem the coefficients and residuals are the dummy regression's exactly. Absorbing costs nothing in available standard error types. `"HC2"` and `"HC3"` are exact at any number of factors, because the leverage of the full design splits into the demeaned-X leverage plus a term that is cheap to compute, so no dummy hat matrix is built. `"CR2"` is the exception: its adjustment is built from cluster-level blocks of the hat matrix rather than from the diagonal, and blocks do not split that way, so it expands the dummies and pays for the expansion. That is why `fixed_effects` with `clusters` defaults to `"CR0"`. Asking for `se_type = "CR2"` still works and still gives the 1.0.6 number. Refused is the three together: `"CR2"` with both `weights` and `fixed_effects`, as in estimatr 1.0.6. The projection identity, the several-factor case, the exact-rank calculation, and the weighted CR2 and HC2 conventions are derived in `vignette("mathematical-notes")`.
se_type	(optional) The standard error type. Defaults depend on whether clusters and/or fixed effects are present: • No clusters, no FE: `"HC2"` (default), `"HC0"`, `"HC1"`, `"HC3"`, `"classical"`, `"stata"`, `"none"`. • Clusters, no FE: `"CR2"` (default), `"CR0"`, `"stata"`, `"none"`. • No clusters, with FE (any number of factors): `"HC2"` (default), `"HC0"`, `"HC1"`, `"HC3"`, `"classical"`, `"stata"`, `"none"`. The same menu as with no FE at all. • Clusters, with FE: `"CR0"` (default), `"CR2"`, `"stata"`, `"none"`. `"CR2"` expands the fixed effects into dummies, so it is not the default here; it is refused with `weights`. `"stata"` means two different things. With no clusters it is exactly `"HC1"`, and the fitted object reports `se_type = "HC1"`. With clusters it is **not** an alias for `"CR0"`: it is CR0 scaled by Stata's finite-sample factor, `(J / (J - 1)) * ((N - 1) / (N - K))` on the variance, and the object reports `se_type = "stata"` to keep the distinction visible.
ci	(optional) Logical. Whether to compute p-values and confidence intervals.
alpha	(optional) The significance level, 0.05 by default.

return_vcov (optional) Logical. Whether to return the vcov matrix.

try_cholesky (optional) Logical. Whether to solve by Cholesky decomposition of X'X rather than by the default pivoted QR. FALSE by default, and worth turning on in most applied settings: about 1.4 times faster at n = 100,000 with two regressors, and 1.7 times faster at n = 200,000 with 60 regressors, where it is 0.15s against 0.25s. The saving is per fit, so it is worth most in a simulation that fits the same design thousands of times.

Rank deficiency is caught on either path. Redundant columns come back as NA exactly as they do from lm() whichever path ran, and a design that is rank deficient falls back to the QR.

Which member of a collinear set is dropped follows lm() as well. Rank is detected with the order-preserving criterion of the LINPACK dqrdc2 routine that lm() uses: the columns are walked left to right and the later column of a dependent pair is the one dropped, so the ordering the model was written in is respected. A caller who needs the dropped set before fitting can therefore reproduce it with qr(X)$pivot on the same design matrix; the coefficients that survive, and the variance read off them, agree with lm() on every rank-deficient design.

Whether it is safe turns on one question, whether two regressors are nearly the same variable. Forming X'X squares the condition number, so the Cholesky path has about twice the rounding error of the QR, and only near-collinearity makes that visible. Differences of scale do not, because the columns are normalized before either decomposition, so a covariate in dollars beside one in years costs nothing. For a treatment indicator, a few covariates, block or cluster dummies, the centered interactions lm_lin() builds, or a factorial, the two paths agree to at least 10 significant digits, which is why difference_in_means() sets it to TRUE internally. Agreement falls to about 3 digits as the scaled condition index reaches 1e6, and the QR fallback takes over above roughly 1e8. Nothing interpretable lives in that range: a design at 1e6 returns a coefficient of 4.8e4 with a standard error of 4.6e4 on a regressor whose true effect is zero. To check a design directly, scale the columns first, since the unscaled condition number of a design in mixed units is large for a reason that does not affect the fit: kappa(sweep(X, 2, sqrt(colSums(X^2)), "/"), exact = TRUE).

Value

An object of class "lm_robust", a list holding the estimate table in coefficients, std.error, df, statistic, p.value, conf.low, conf.high, term, and outcome; the fit in fitted.values, residuals, vcov, nobs, k, rank, df.residual, and res_var; the summary statistics r.squared, adj.r.squared, tss, and fstatistic; and se_type, weighted, clustered, fes, alpha, terms, xlevels, and call. Under se_type = "HC2" or "HC3", n_leverage_near_one counts the observations at or near leverage 1 that the variance discards; it is 0 for every other se_type.

Absorbed fits add fixed_effects, felevels (the absorbed levels of each factor), and the within-projection summaries proj_r.squared, proj_adj.r.squared, proj_tss, and proj_fstatistic.

Examples

```
set.seed(15)
```

```
dat <- data.frame(
  y = rpois(40, lambda = 4),
  x = rnorm(40),
  z = rbinom(40, 1, prob = 0.4),
  cl = rep(1:10, each = 4),
  bl = rep(c("A", "B", "C", "D"), each = 10),
  w = runif(40)
)

# HC2 is the default
fit <- lm_robust(y ~ x + z, data = dat)
fit
tidy(fit)
summary(fit)
confint(fit, level = 0.8)

# Other variance estimators, including Stata's
lm_robust(y ~ x + z, data = dat, se_type = "classical")
lm_robust(y ~ x + z, data = dat, se_type = "stata")

# Clustered inference defaults to CR2
lm_robust(y ~ x + z, data = dat, clusters = cl)
lm_robust(y ~ x + z, data = dat, clusters = cl, se_type = "stata")

# Weights and subsets behave as they do in lm()
lm_robust(y ~ x + z, data = dat, weights = w, clusters = cl)
lm_robust(y ~ x, data = dat, subset = z == 1)

# Fixed effects are absorbed rather than expanded into dummies. With a
# single factor the HC2 default is exact and costs nothing extra.
lm_robust(y ~ z, data = dat, fixed_effects = ~ bl)
```

`lm_robust_fit` *Internal method that creates linear fits*

Description

Internal method that creates linear fits

Usage

```
lm_robust_fit(
  y,
  X,
  weights,
  cluster,
  ci = TRUE,
  se_type,
  has_int,
```

```
  alpha = 0.05,
  return_vcov = TRUE,
  return_fit = TRUE,
  try_cholesky = FALSE,
  iv_stage = list(0),
  fe_rank = 0L,
  fe_leverage = NULL,
  femat = NULL,
  linear_hypothesis = NULL
)
```

Arguments

y	numeric outcome vector or matrix
X	numeric design matrix
weights	numeric weights vector
cluster	numeric cluster vector
ci	boolean, whether to return confidence intervals and p-values
se_type	character denoting which kind of SEs to return
has_int	logical, whether the model has an intercept
alpha	numeric, test size for confidence intervals
return_vcov	logical, whether to return the vcov matrix
return_fit	logical, whether to return fitted values
try_cholesky	logical. Solve by Cholesky decomposition of X'X rather than by the pivoted QR, falling back to the QR where the design is rank deficient. See lm_robust() for when the fast path is safe.
iv_stage	list of length one or two for 2SLS stages
fe_rank	integer, degrees of freedom absorbed by fixed effects
fe_leverage	numeric vector of per-observation leverage contributed by the absorbed fixed effects, or NULL. h_ii of the full design splits exactly into the demeaned-X leverage plus this term, for any number of FE factors, which is what makes HC2 and HC3 available under fixed_effects without building the dummy matrix.
femat	optional numeric matrix of fixed-effect dummies for the estimation sample. Only CR2 needs it: its adjustment is built from cluster-level blocks of the hat matrix, which do not decompose the way the diagonal does. HC2 and HC3 take fe_leverage instead. NULL unless the requested se_type requires it.
linear_hypothesis	optional hypotheses, in the form lh_robust() takes, whose CR2 Satterthwaite degrees of freedom are returned as hypothesis_df. Ignored for every other se_type, where a combination of coefficients has the same degrees of freedom as each of them.

Examples

```
# The fitter behind lm_robust(), exported for packages that have already
# built their own design matrix. Most users want lm_robust().
set.seed(45)
X <- cbind(`(Intercept)` = 1, x = rnorm(50))
y <- X[, "x"] + rnorm(50)

lm_robust_fit(
  y = y, X = X,
  weights = NULL, cluster = NULL,
  se_type = "HC2", has_int = TRUE
)
```

---

predict.lm_robust *Predict method for* lm_robust *object*

Description

Produces predicted values, obtained by evaluating the regression function in the frame newdata for fits from lm_robust() and lm_lin(). If se.fit is TRUE, standard errors of the predictions are calculated. Setting interval adds confidence or prediction (tolerance) intervals at the level set by alpha, sometimes called narrow and wide intervals respectively.

Usage

```
## S3 method for class 'lm_robust'
predict(
  object,
  newdata,
  se.fit = FALSE,
  interval = c("none", "confidence", "prediction"),
  alpha = 0.05,
  na.action = na.pass,
  pred.var = NULL,
  weights,
  ...
)
```

Arguments

object	An object of class "lm_robust".
newdata	A data frame in which to look for the variables to predict from. If omitted, the fitted values are returned.
se.fit	Logical. Whether to return standard errors. FALSE by default.
interval	Type of interval calculation, which can be abbreviated. "none" by default.

alpha	Numeric. The test size for confidence intervals.
na.action	Function determining what to do with missing values in newdata. The default is to predict NA.
pred.var	The variance(s) to assume for future observations when building prediction intervals.
weights	Variance weights for prediction, either a numeric vector or the bare (unquoted) name of the weights variable in newdata.
...	(optional) Ignored.

Details

Called without newdata, the method returns the in-sample fitted values, and neither se.fit nor interval is available.

The equation used for the standard error of a prediction given a row of data x is:

$\sqrt{(x \Sigma x')}$,

where Σ is the estimated variance-covariance matrix from lm_robust().

The prediction intervals are for a single observation at each case in newdata with error variance(s) pred.var. The default is to assume that future observations have the same error variance as those used for fitting, which is taken from the fitted lm_robust() object. If weights is supplied, the inverse of those weights scales the variance. If the fit was weighted, the default is to assume constant prediction variance, with a warning.

Value

A numeric vector of predictions, or a data frame with the predictions and their standard errors and interval bounds when se.fit or interval is set.

See Also

lm_robust(), lm_lin()

Examples

```
# Set seed
set.seed(42)

# Simulate data
n <- 10
dat <- data.frame(y = rnorm(n), x = rnorm(n))

# Fit lm
lm_out <- lm_robust(y ~ x, data = dat)
# In-sample fitted values
predict(lm_out)
# Get predicted fits
fits <- predict(lm_out, newdata = dat)
# With standard errors and confidence intervals
fits <- predict(lm_out, newdata = dat, se.fit = TRUE, interval = "confidence")
```

```
# Use new data as well
new_dat <- data.frame(x = runif(n, 5, 8))
predict(lm_out, newdata = new_dat)

# You can also supply custom variance weights for prediction intervals
new_dat$w <- runif(n)
predict(lm_out, newdata = new_dat, weights = w, interval = "prediction")

# Works for 'lm_lin' models as well
dat$z <- sample(1:3, size = nrow(dat), replace = TRUE)
lmlin_out1 <- lm_lin(y ~ z, covariates = ~ x, data = dat)
predict(lmlin_out1, newdata = dat, interval = "prediction")
```

Index

augment.iv_robust *(*augment.lm_robust*)*, 3
augment.lm_robust, 3

broom::augment(), *3*

commarobust *(*estimatr-defunct*)*, 6

difference_in_means, 4
difference_in_means(), *14*, *17*, *20*

estimatr *(*estimatr-package*)*, 2
estimatr-defunct, 6
estimatr-package, 2
estimatr_tidiers, 7
extract.iv_robust *(*extract.lm_robust*)*, 8
extract.lm_robust, 8

glance(), *7*, *9*

horvitz_thompson, 10

iv_robust, 12

lh_robust, 14
lh_robust(), *22*
lm(), *13*, *16*, *19*, *20*
lm_lin, 15
lm_lin(), *13*, *17*, *20*, *23*, *24*
lm_robust, 18
lm_robust(), *5*, *7*, *13–17*, *22–24*
lm_robust_fit, 21

predict.lm_robust, 23

starprep *(*estimatr-defunct*)*, 6

tidy(), *7*, *9*
tidy.difference_in_means
        *(*estimatr_tidiers*)*, 7
tidy.horvitz_thompson
        *(*estimatr_tidiers*)*, 7
tidy.iv_robust *(*estimatr_tidiers*)*, 7
tidy.lh *(*estimatr_tidiers*)*, 7
tidy.lh_robust *(*estimatr_tidiers*)*, 7
tidy.lm_robust *(*estimatr_tidiers*)*, 7